Attorney Docket No. MRNRP002

PATENT APPLICATION

TECHNIQUE FOR ASSIGNING SCHEDULE RESOURCES TO MULTIPLE PORTS IN CORRECT PROPORTIONS

Inventors:

Kenneth W Brinkerhoff
27825 Perales
Mission Viejo, CA 92692
Citizen of U.S.A.

Wayne P Boese
2053A Tustin Ave.
Costa Mesa, CA 92627
Citizen of U.S.A.

Robert C Hutchins
24272 Solonica St.
Mission Viejo, CA 92691
Citizen of U.S.A.

Stanley Wong
2409 West Hall Ave.
Santa Ana, CA 92704
Citizen of U.S.A.

Assignee:

Mariner Networks, Inc.
1585 S. Manchester Avenue
Anaheim, CA 92802-2907

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
Telephone (510) 843-6200

TECHNIQUE FOR ASSIGNING SCHEDULE RESOURCES TO MULTIPLE PORTS IN CORRECT PROPORTIONS

Inventors:

Kenneth W Brinkerhoff
27825 Perales
Mission Viejo, CA 92692
Citizen of U.S.A.

Wayne P Boese
2053A Tustin Ave.
Costa Mesa, CA 92627
Citizen of U.S.A.

Robert C Hutchins
24272 Solonica St.
Mission Viejo, CA 92691
Citizen of U.S.A.

Stanley Wong
2409 West Hall Ave.
Santa Ana, CA 92704
Citizen of U.S.A.

Assignee:

Mariner Networks, Inc.
1585 S. Manchester Avenue
Anaheim, CA 92802-2907

RELATED APPLICATION DATA

The present application claims priority under 35 USC 119(e) from U.S. Provisional Patent Application No. 60/215,558 (Attorney Docket No. MO15-1001-Prov) entitled "INTEGRATED ACCESS DEVICE FOR ASYNCHRONOUS TRANSFER MODE (ATM) COMMUNICATIONS"; filed June 30, 2000, and naming Brinkerhoff, et. al., as inventors (attached hereto as Appendix A); the entirety of which is incorporated herein by reference for all purposes.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to data networks, and more specifically to a technique for implementing fractional interval times for fine granularity bandwidth allocation.

Description of Related Arts

The dynamic variation of port speed in high bandwidth switches is a relatively new phenomenon. Analog modems have long had the feature of connecting at different bit rates, some of them even changing bit rates during a session. However, such modems typically operate over a standard 64 Kb voice channel through the network, and adapt to changes measured in the signal to noise ratio of given connection by changing the modulation scheme used. The recent proliferation of ATM Bandwidth on Demand and Rate Adaptive DSL, however, have injected a new characteristic into high speed switching, namely, high speed ports whose line rates change dynamically.

The scheduling of data parcels to an output or egress port requires some amount of scheduler resources to be allocated for each data parcel that is processed. Accordingly, many switch implementations face the problem of efficiently managing the scheduler resources. For example, where throughput performance is a primary consideration, a dedicated scheduler may be provided for each port so each port has immediate access to scheduling resources. However, when economics is a primary consideration, scheduling resources are typically shared among multiple ports.

It becomes more complicated when a scheduling resource is shared across ports with different line rates. For example, in conventional communication systems which have multiple client flows, each client flow may have a corresponding relative need of service. The frequency of service for a particular client flow (typically associated with a respective port) is related to the number of data parcels (e.g. cells, frames, packets, etc.) which need to be serviced per second from the port associated with that client flow. This concept holds true for any type of time-dependent service in which a service entity sponsors multiple client flows each having time-sensitive needs. Some clients may have relaxed needs, while other clients may have very stringent time sensitive needs.

Typically, under such conditions, issues arise as to how to properly sequence the order in which each of the client flows are serviced so that the clients which need to be serviced frequently are serviced sufficiently frequently, and other clients which require less frequent service are also serviced sufficiently frequently, but not to the point where service needs are taken away from other client flows which have more stringent needs. Thus, the scheduling algorithm must balance the frequency of needs between all clients against the scheduling resources available.

FIGURE 1A shows an example of a service entity 100 which has enabled a plurality of client flows 101. As shown in the example of FIGURE 1A, each client or service flow has a respective service need, which, for example, may be associated with a respective bit rate for each client flow. For example, Client 101A requires that 4 megabits of its data be serviced every second, corresponding to a bit rate of 4 Mbps; Client 101B requires that 8 megabits of its data be serviced every second, corresponding to a bit rate of 8 Mbps; and Client 101C requires that 2 megabits of its data be serviced every second corresponding to a bit rate of 2 Mbps. Thus, it will be appreciated that one issue which arises for servicing the plurality of client flows 101 is how to service each of the client flows adequately without affecting the service needs of the other client flows.

Traditional algorithms for sharing scheduler resources vary from simple round-robin port service to weighted service based on an ordered list of port numbers. Weighted priority works well when the port line rates are constant. Round-robin service works well when ports are approximately equal in line rate. In conventional

round-robin scheduling protocols, each client flow is serviced in a particular order using a static scheduling list such as that, for example, shown in FIGURE 1B.

FIGURE 1B shows an example of a scheduling list 150 which may be used to perform round-robin scheduling for multiple client flows. According to conventional techniques, the scheduling list 150 is populated according to the relative service needs of each client flow. For example, referring to FIGURE 1A, Client 1 requires servicing twice as often as Client 3, and Client 2 requires servicing 4 times as often as Client 3. Accordingly, as shown in FIGURE 1B, a scheduling list may be generated in which Client 2 is serviced 4 times as often as Client 3, and in which Client 1 is serviced twice as often as Client 3. Once this scheduling list has been generated, the scheduler may then service each client flow by walking down the service list, and servicing each client flow in the order in which it appears in the service list. Thus, for example, Client 2 may be serviced twice, followed by Client 1 being serviced once, followed by Client 2 being serviced twice, followed by Client 1 being serviced once, followed by Client 3 being serviced once. After Client 3 has been serviced, the scheduler may then return to the top of the list and repeat this servicing order once again.

It will be appreciated that efficient utilization of scheduling resources becomes even more complicated in situations where the client ports sharing the scheduling resource have dynamically changing bandwidth needs. Currently none of the traditional scheduling algorithms provide adequate performance for servicing port line rates which vary dynamically. Accordingly, it will be appreciated that continual research efforts are being undertaken to improve multiple port resource scheduling techniques.

SUMMARY OF THE INVENTION

According to specific embodiments, the present invention describes a new technique for providing service to multiple ports sharing common scheduling resources. According to one implementation, the scheduling technique of the present invention may be used to dynamically balance the frequency of needs of different client flows to the resource availability of the scheduling process for client flows which have relative time sensitive needs of service. Moreover, according to a specific implementations, the scheduling technique of the present invention may be used to provide efficient

allocation of switching and/or scheduling resources across multiple ports even in the presence of dynamic port bandwidth changes.

Alternate embodiments of the present invention are directed to methods, computer program products, and systems for scheduling service of traffic relating to a plurality of different communication flows. Each communication flow may have a respective service need associated therewith. A first service order is determined for servicing the plurality of communication flows, the first service order being based upon the relative service needs of each of the plurality of communication flows. When a change in the service needs of one or more communication flows is detected, a new service order may be automatically and/or dynamically determined for servicing the plurality of communication flows. According to a specific implementation, the new service order may be based upon the relative service needs of each of the plurality of communication flows, including the new service needs corresponding any of the communication flows.

According to specific embodiments, the dynamic scheduling technique of the present invention utilizes one or more data structures having entries which correspond to specific communication flows. Each entry may include a respective communication flow identifier (e.g. port ID) and a respective time key value. The data structure may be ordered so that the next communication flow to be serviced is located at a specified position in the data structure. After the next selected communication flow has been serviced, the time key value for that communication flow may be updated and a next entry for the communication flow reinserted into the data structure appropriately to queue up its next service commensurate with that communication flow's particular communication rate. According to one implementation of the present invention, the time key parameter may be defined such that any changes in a communication flow's communication rate may automatically be taken into account the next time that communication flow is serviced.

Additional objects, features and advantages of the various aspects of the present invention will become apparent from the following description of its preferred embodiments, which description should be taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1A shows an example of a service entity 100 which has enabled a plurality of client flows 101.

FIGURE 1B shows an example of a scheduling list 150 which may be used to perform round-robin scheduling for multiple client flows.

FIGURE 2 shows a flow diagram of a Dynamic Scheduling Procedure 200 in accordance with a specific embodiment of the present invention.

FIGURE 4 shows a flow diagram of a Service Need Monitoring Process 400 in accordance with a specific embodiment of the present invention.

FIGURE 5 shows an example of a portion of a customer entity system 500, which may be used for illustrating various aspects of the present invention.

FIGURE 6 shows an example of a Dynamic Scheduling Data Structure 600 in accordance with a specific embodiment of the present invention.

FIGURE 7 shows a specific embodiment of a network device 60 suitable for implementing various techniques of the present invention.

FIGURE 8 shows a block diagram of various inputs/outputs, components and connections of a system 800 which may be used for implementing various aspects of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

According to specific embodiments, the present invention describes a new technique for providing service to multiple ports sharing common scheduling resources. According to one implementation, the scheduling technique of the present invention may be used to perform QoS scheduling of data parcels for one or multiple ports. Depending upon performance and costs requirements, the technique of the present invention may be implemented in hardware and/or software.

According to specific embodiments, the dynamic scheduling technique of the present invention utilizes a stack of entries corresponding to specific client flows, wherein each entry includes a client flow identifier (e.g. port ID) and a pseudo time key value. The stack may be ordered so that the next client flow to be serviced is located at a specified position in the stack (e.g. at the front of the stack). After the next selected client flow has been serviced, the pseudo time key value for that client flow may be

updated and a next entry for the client flow reinserted into the stack appropriately to queue up its next service commensurate with that client flow's particular line rate. According to one implementation of the present invention, any changes in a client flow's line rate may automatically be taken into account the next time that client flow is serviced.

It will be appreciated that the technique of the present invention may be used to balance the frequency of needs of different client flows to the resource availability of the scheduling process for client flows which have relative time sensitive needs of service. Moreover, the scheduling technique of the present invention results in efficient allocation of switching and/or scheduling resources across multiple ports even in the presence of dynamic port bandwidth changes, with little or no added costs.

FIGURE 5 shows an example of a portion of a customer entity system 500, which may be used for illustrating various aspects of the present invention. In the example of FIGURE 5, it is assumed that the client flows C1-C3 each have relatively different service need requirements, which have been described in terms of the number of bits per second associated with each respective flow. For example, Client C1 requires that 4 megabits of its data be serviced every second, corresponding to a bit rate of 4 Mbps; Client C2 requires that 8 megabits of its data be serviced every second, corresponding to a bit rate of 8 Mbps; and Client C3 requires that 2 megabits of its data be serviced every second corresponding to a bit rate of 2 Mbps.

According to a specific embodiment, a client interval value or client interval parameter I_I may be used to describe the rate at which data parcels are to be serviced for a particular client flow, where the variable [i] may be used to denote a specific client flow. For example, according to one implementation, the value [i] may correspond to a port number associated with a specific client flow. In one implementation, the client interval parameter I_I for a selected client flow may be directly related to the bit rate associated with that client flow. Alternatively, according to a different implementation, the client interval parameter I_I for a selected client flow may be inversely related to the bit rate associated with that client flow. Thus, for example, in this latter implementation, the greater the service need requirements a client flow, the lower it's associated I_I value will be. According to a specific implementation, an external process

may be used to determine and keep track of service timing needs for each client flow, including the calculation of the I_I value for each client flow.

According to a specific embodiment, the client interval values I_I for one or more client flows may be determined based upon the respective bit rate for each client flow. In the present application, the bit rate associated with a given client [i] may be represented by the variable R_I. According to one implementation, the client interval parameter I_I may be described as:

$$I_I = \frac{RANGE}{R_I}, \qquad (1)$$

where the parameter RANGE is defined as:

$$RANGE = k * \sum_{i=1}^{n} R_I, \qquad (2)$$

for client flows 1 through n, where the variable k may be used to define a desired granularity factor (e.g. k = 10).

Thus, it will be appreciated from the definitions above that client flows having relatively larger service flow needs will have relatively lower client interval values. The smaller the client interval value, the greater the service needs of the client flow associated with the client interval value.

The value R_I associated with a particular client flow may correspond to a bit rate of that client flow, or a rate in which data parcels are to be serviced from that particular client (e.g. frames/cells per second). In determining the I_I value, it is preferable that the R_I values be expressed in terms of normalized units.

As shown in the example of FIGURE 5, client flow C1 has an associated service rate need of R1 = 4 Mbps, client flow C2 has an associated service rate need of R2 = 8 Mbps, and client flow C3 has an associated service rate need of R3 = 2 Mbps.

According to a specific embodiment, in order to calculate the client interval value I_I for each client flow of FIGURE 5, a range of numbers is used to meaningfully represent the relative line rates and service times for each client flow. The upper end of the range is defined by adding each of the individual R_I values for each client flow at their maximum line rates, and then multiplying this value by a number (k) sufficiently large to expand it by an order of magnitude such as, for example, by 10 or 16. This

increased range may then be used to provide good granularity in representing relative line rates and intervals.

Thus, referring to the example of FIGURE 5, a RANGE parameter is first determined by adding the maximum line rates of each client flow (e.g. $4 + 8 + 2 = 14$). This sum is then multiplied by a granularity factor k. In the present example, it is assumed that $k = 16$, which results in the new RANGE value being equal to 224. According to a specific embodiment, this result may be rounded up to an even power of 2, if desired, for convenience in calculations. Alternatively, according to a specific implementation, the RANGE value may be rounded up to a value which is evenly divisible by all R_I values. In either case, the resulting value may be stored as a variable named RANGE. In the present example, it is assumed that the variable RANGE = 224.

Once the RANGE value has been determined, individual client interval values may then be calculated based upon the equation $I_I = RANGE / R_I$. Thus, for example, the client interval value I1 for client C1 may be calculated as I1 = RANGE/R1 = 224/4 = 56. The client interval value I2 for client C2 may be calculated as I2 = RANGE/R2 = 224/8 = 28. The client interval value I3 for client C3 may be calculated as I3 = RANGE/R3 = 224/2 = 112.

Using the example values as shown in FIGURE 5, the Dynamic Scheduling Procedure of FIGURE 2 will now be described in greater detail.

FIGURE 2 shows a flow diagram of a Dynamic Scheduling Procedure 200 in accordance with a specific embodiment of the present invention. According to a specific implementation, the Dynamic Scheduling Procedure 200 may be implemented at the scheduler 507 of FIGURE 5. For purposes of illustration, it will be assumed that the Dynamic Scheduling Procedure of FIGURE 2 is being implemented on the system 500 of FIGURE 5.

According to a specific embodiment of the present invention, the scheduler 507 may be configured to sequentially service multiple client flows whose data is to be received via one or more client ports. The scheduler may be configured to dynamically determine which client port to service next based upon the relative service needs of each client flow. For example, client ports with higher line rates may need to be serviced by the scheduler more frequently than client ports with lower line rates. According to specific embodiments, the technique of the present invention provides an

algorithm for automatically and dynamically determining the sequence in which ports are serviced by the scheduler to assure that each port receives the proportion and/or frequency of service commensurate with its associated line rate.

Initially, as shown at 202, a respective variable N_I may be initialized for each or selected client flows, and set equal to a particular value. In the embodiment of FIGURE 2, the value N_I for a particular client [i] may be set equal to the I_I value for that client. Thus, for example, the variable N1 may be set equal to N1 = I1 = 56, the variable N2 may be set equal to N2 = I2 = 28, and the variable N3 may be set equal to N3 = I3 = 112.

According to a specific embodiment, each of the variables N_I and I_I for selected client flows may be populated and stored in one or more data structures such as that shown, for example, in FIGURE 3 of the drawings.

FIGURE 3 shows a specific embodiment of an entry 300 which may be stored in a Dynamic Scheduling Data Structure used, for example, for implementing the Dynamic Scheduling Procedure 200 of FIGURE 2. It will be appreciated that the Dynamic Scheduling Data Structure may include a plurality of entries, wherein each entry corresponds to a different client flow. As shown in the example of FIGURE 3, each entry 300 includes a first field 302 which may be used for identifying a specific client flow, a second field 304 which may be used for storing the current N_I value associated with that client flow, and a third field 306 which may be used for storing the current I_I value associated with that client flow. According to specific embodiments, the variables N_I and I_I may be defined as an n-bit binary number, where n may be determined based upon specific implementations.

According to one implementation, the Dynamic Scheduling Data Structure may be indexed by the Client Flow ID field 302. For example, in one implementation, the Dynamic Scheduling Data Structure may be indexed by port number, wherein each port number is associated with a specific client flow.

Additionally, as shown in FIGURE 3, a KEY_I parameter or value for each client flow may also be defined. According to a specific implementation, the KEY_I value for a specific client flow may be defined by the equation:

$$KEY_I = N_I \mid I_I, \qquad (3)$$

where the symbol "|" represents a concatenation operation. Thus, according to a specific embodiment, the fields 304 and 306 of FIGURE 3 may be concatenated together to represent the KEY_I value 308 corresponding to that particular client flow. For example, if the variable N1 is represented as a binary integer having the value 00001001, and the variable I1 is represented as a binary integer having the value 00000100, then the KEY1 value corresponding to this client flow may automatically be defined as 0000100100000100. Thus, according to a specific embodiment, as shown in FIGURE 3, the N_I value for each client flow represents the most significant bits of the KEY_I value associated with that client, and the I_I value for that client flow represents the least significant bits of the KEY_I value associated with that client flow.

According to specific embodiments, a concept of pseudo time is used to provide a relative numeric indicator of temporal need for service for selected client flows. In specific embodiments, the relative numeric indicator of temporal need for service of each selected client flow may be expressed by the KEY_I value associated with each client flow.

FIGURE 6 shows an example of a Dynamic Scheduling Data Structure 600 in accordance with a specific embodiment of the present invention. For purposes of explanation, the Dynamic Scheduling Data Structure 600 of FIGURE 6 shows a snapshot of values which may be stored in the Data Structure 600 after operation 202 of FIGURE 2 has been successfully completed.

According to a specific embodiment, the Dynamic Scheduling Data Structure may be implemented as a service request stack, wherein the entries of the service request stack are ordered based upon their respective KEY_I values. In one implementation, entries with the relatively lowest KEY_I values are placed at the front of the stack. Each time an entry in the service request stack is updated, the stack may then be re-sorted so that the entry with the smallest KEY_I value is at the front of the stack. It will be appreciated that there are a variety of different sorting methods which may be employed to accomplish comparison and location of KEY_I values in the stack. Example of such sorting methods include binary search and balanced tree indexing. Once the entries of the service request stack have been sorted according to their respective KEY_I values, the scheduler may then select the front service request entry of the stack, and service the client flow associated with the selected entry.

Returning to FIGURE 2, upon completion of operation 202, the Dynamic Scheduling Data Structure may be populated as follows:

Client Flow ID	**N_I Value**	**I_I Value**
C1	056	056
C2	028	028
C3	112	112

It will be appreciated that, according to a specific embodiment, the KEY_I value for each client flow may be automatically and/or implicitly determined using the values from the Dynamic Scheduling Data Structure, without having to perform additional computation operations. The scheduler may then identify and select (204) a specific client flow having the smallest KEY_I value.

As shown in the Dynamic Scheduling Data Structure (above), the client flow with the smallest KEY_I value corresponds to client flow C2, which has an associated KEY_I value of KEY2 = 028028.

The scheduler may then service (206) the selected client flow (e.g. client flow C2). Thereafter, the N_I value for the selected client flow is incremented or updated. In the specific embodiment of FIGURE 2, the N_I value of the selected client flow is incremented by the I_I value associated with that client flow. Thus, in the present example, the value N2 will be incremented to N2 = 028 + I3 = 028 + 028 = 056.

It will be appreciated that, each time a particular N_I is incremented, the corresponding KEY_I value associated with that client flow will automatically be updated to reflect a new value. Moreover, according to specific embodiments of the present invention, the automatic updating of the KEY_I may be accomplished merely by updating the N_I value and/or value or the I_I value for a particular client, and does not necessarily require additional processing steps in order to compute or calculate the new KEY_I value, since the KEY_I value is defined as $KEY_I = N_I \mid I_I$.

After the value for N2 has been updated to N2 = 056, the update Dynamic Scheduling Data Structure may be populated as follows:

Client Flow ID	N_I Value	I_I Value
C1	056	056
C2	056	028
C3	112	112

Thereafter, the Dynamic Scheduling Procedure identifies and selects (204) a new client flow having the smallest KEY_I value.

In the current example, the scheduler will again select client flow C2 since its KEY_I value of KEY2 = 056028 is the smallest KEY_I value in the Dynamic Scheduling Data Structure. Accordingly, the scheduler will service (206) client flow C2, and then increment (208) the N2 value to N2 = 056 + 028 = 084.

At the next iteration of the Dynamic Scheduling Procedure, the Dynamic Scheduling Data Structure may be populated as follows:

Client Flow ID	N_I Value	I_I Value
C1	056	056
C2	084	028
C3	112	112

The Dynamic Scheduling Procedure then identifies and selects (204) a new client flow having the smallest KEY_I value.

In the current example, the scheduler will select client flow C1 since its KEY_I value of KEY1 = 056056 is the smallest KEY_I value in the Dynamic Scheduling Data Structure. Accordingly, the scheduler will service (206) client flow C, and then increment (208) the N1 value to N1 = 056 + 0056 = 112.

It will be appreciated that, with each iteration of the Dynamic Scheduling Procedure, a next client flow may be selected and serviced by the scheduler 507.

As stated previously, an external process may be used to keep track of service timing needs for each client flow. If the service needs of one or more client flows should change, either statically or dynamically, the external process may recalculate the I_I value(s) for each client flow having a change in service needs. Once the updated I_I value has been calculated, the external process may then store the new I_I value in the Dynamic Scheduling Data Structure. This is shown for example, in FIGURE 4 of the drawings.

FIGURE 4 shows a flow diagram of a Service Need Monitoring Process 400 in accordance with a specific embodiment of the present invention. According to a specific implementation, the Service Need Monitoring Process 400 may be implemented as an external process to the scheduler which monitors the service needs of each client flow, and updates the I_I values for each client flow, dynamically, as need.

For example, as shown at 402 of FIGURE 4, the Service Need Monitoring Process monitors each client flow to determine whether any client flow service needs have changed. Examples of changing service needs include increased or decreased bandwidth demands, increased or decreased line rate requirements, etc. Assuming that at least one client flow has been identified (404) as having changed service needs, the Service Need Monitoring Process calculates and updates (406) the I_I value(s) for the identified client flow(s).

According to a specific implementation, the updated I_I value(s) are automatically stored in the Dynamic Scheduling Data Structure such as that shown in FIGURE 6. It will be appreciated that, according to a specific embodiment, the updating of any I_I value in the Dynamic Scheduling Data Structure will automatically result in an updated KEY_I value for that particular entry. Moreover, the updated KEY_I value will automatically be factored in the next time the Dynamic Scheduling Procedure selects (204) a next client flow having the smallest key I_I value.

Thus, for example, when port rate changes occur, a process associated with managing the port will update the I_I value associated with that port in the Dynamic Scheduling Data Structure, thereby resulting in an updated KEY_I value associated with that port which will automatically be factored into future port servicing sequencing.

It will be appreciated that the scheduling technique of the present invention provides a mechanism for dynamically servicing multiple client flows having different service needs in an automated manner which is able to adapt to dynamically changing service needs of each client flow. Moreover, using the technique of the present invention, a single service list may be generated for multiple client flows having different service needs. Additionally, according to a specific embodiment, the updating of the service needs of any client flow may be handled dynamically without using multiple different service lists, and without interrupting service to accommodate the new service need requirements.

System Configurations

Referring now to FIGURE 7, a network device 60 suitable for implementing the scheduling resource allocation techniques of the present invention includes a master central processing unit (CPU) 62A, interfaces 68, and various buses 67A, 67B, 67C, etc., among other components. According to a specific implementation, the CPU 62A may correspond to the eXpedite ASIC, manufactured by Mariner Networks, of Anaheim, California.

Network device 60 is capable of handling multiple interfaces, media and protocols. In a specific embodiment, network device 60 uses a combination of software and hardware components (e.g., FPGA logic, ASICs, etc.) to achieve high-bandwidth performance and throughput (e.g., greater than 6 Mbps), while additionally providing a high number of features generally unattainable with devices that are predominantly either software or hardware driven. In other embodiments, network device 60 can be implemented primarily in hardware, or be primarily software driven.

When acting under the control of appropriate software or firmware, CPU 62A may be responsible for implementing specific functions associated with the functions of a desired network device, for example a fiber optic switch or an edge router. In another example, when configured as a multi-interface, protocol and media network device, CPU 62A may be responsible for analyzing, encapsulating, or forwarding packets to appropriate network devices. Network device 60 can also include additional processors or CPUs, illustrated, for example, in FIGURE 7 by CPU 62B and CPU 62C. In one implementation, CPU 62B can be a general purpose processor for handling network management, configuration of line cards, FPGA logic configurations, user interface configurations, etc. According to a specific implementation, the CPU 62B may correspond to a HELIUM Processor, manufactured by Virata Corp. of Santa Clara, California. In a different embodiment, such tasks may be handled by CPU62A, which preferably accomplishes all these functions under partial control of software (e.g., applications software and operating systems) and partial control of hardware.

CPU 62A may include one or more processors 63 such as the MIPS, Power PC or ARM processors. In an alternative embodiment, processor 63 is specially designed hardware (e.g., FPGA logic, ASIC) for controlling the operations of network device 60. In a specific embodiment, a memory 61 (such as non-persistent RAM and/or ROM)

also forms part of CPU 62A. However, there are many different ways in which memory could be coupled to the system. Memory block 61 may be used for a variety of purposes such as, for example, caching and/or storing data, programming instructions, etc.

According to a specific embodiment, interfaces 68 may be implemented as interface cards, also referred to as line cards. Generally, the interfaces control the sending and receiving of data packets over the network and sometimes support other peripherals used with network device 60. Examples of the interfaces that may be provided are Ethernet interfaces, frame relay interfaces, cable interfaces, DSL interfaces, token ring interfaces, IP interfaces, etc. In addition, various ultra high-speed interfaces can be provided such as fast Ethernet interfaces, Gigabit Ethernet interfaces, ATM interfaces, HSSI interfaces, POS interfaces, FDDI interfaces and the like. Generally, these interfaces include ports appropriate for communication with appropriate media. In some cases, they also include an independent processor and, in some instances, volatile RAM. The independent processors may control communications intensive tasks such as data parcel switching, media control and management, framing, interworking, protocol conversion, data parsing, etc. By providing separate processors for communications-intensive tasks, these interfaces allow the main CPU 62A to efficiently perform routing computations, network diagnostics, security functions, etc. Alternatively, CPU 62A may be configured to perform at least a portion of the above-described functions such as, for example, data forwarding, communication protocol and format conversion, interworking, framing, data parsing, etc.

In a specific embodiment, network device 60 is configured to accommodate a plurality of line cards 70. At least a portion of the line cards are implemented as hot-swappable modules or ports. Other line cards may provide ports for communicating with the general-purpose processor, and may be configured to support standardized communication protocols such as, for example, Ethernet or DSL. Additionally, according to one implementation, at least a portion of the line cards may be configured to support Utopia and/or TDM connections.

Although the system shown in FIGURE 7 illustrates one specific network device of the present invention, it is by no means the only network device architecture

on which the present invention can be implemented. For example, an architecture having a single processor that handles communications as well as routing computations, etc., may be used. Further, other types of interfaces and media could also be used with the network device such as TI, E1, Ethernet or Frame Relay.

According to a specific embodiment, network device 60 may be configured to support a variety of different types of connections between the various components. For illustrative purposes, it will be assumed that CPU 62A is used as a primary reference component in device 60. However, it will be understood that the various connection types and configurations described below may be applied to any connection between any of the components described herein.

According to a specific implementation, CPU 62A supports connections to a plurality of Utopia lines. As commonly known to one having ordinary skill in the art, a Utopia connection is typically implemented as an 8-bit connection which supports standardized ATM protocol. In a specific embodiment, the CPU 62A may be connected to one or more line cards 70 via Utopia bus 67A and ports 69. In an alternate embodiment, the CPU 62A may be connected to one or more line cards 70 via point-to-point connections 51 and ports 69. The CPU 62A may also be connected to additional processors (e.g. 62B, 62C) via a bus or point-to-point connections (not shown). As described in greater detail below, the point-to-point connections may be configured to support a variety of communication protocols including, for example, Utopia, TDM, etc.

As shown in the embodiment of FIGURE 7, CPU 62A may also be configured to support at least one bi-directional Time-Division Multiplexing (TDM) protocol connection to one or more line cards 70. Such a connection may be implemented using a TDM bus 67B, or may be implemented using a point-to-point link 51.

In a specific embodiment, CPU 62A may be configured to communicate with a daughter card (not shown) which can be used for functions such as voice processing, encryption, or other functions performed by line cards 70. According to a specific implementation, the communication link between the CPU 62A and the daughter card may be implemented using a bi-directional TDM connection and/or a Utopia connection.

According to a specific implementation, CPU 62B may also be configured to communicate with one or more line cards 70 via at least one type connection. For example, one connection may include a CPU interface that allows configuration data to be sent from CPU 62B to configuration registers on selected line cards 70. Another connection may include, for example, an EEPROM arrow interface to an EEPROM memory 72 residing on selected line cards 70.

Additionally, according to a specific embodiment, one or more CPUs may be connected to memories or memory modules 65. The memories or memory modules may be configured to store program instructions and application programming data for the network operations and other functions of the present invention described herein. The program instructions may specify an operating system and one or more applications, for example. Such memory or memories may also be configured to store configuration data for configuring system components, data structures, or other specific non-program information described herein.

Because such information and program instructions may be employed to implement the systems/methods described herein, the present invention relates to machine-readable media that include program instructions, state information, etc. for performing various operations described herein. Examples of machine-readable media include, but are not limited to, magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as CD-ROM disks; magneto-optical media such as floptical disks; and hardware devices that are specially configured to store and perform program instructions, such as read-only memory devices (ROM), Flash memory PROMS, random access memory (RAM), etc.

In a specific embodiment, CPU 62B may also be adapted to configure various system components including line cards 70 and/or memory or registers associated with CPU 62A. CPU 62B may also be configured to create and extinguish connections between network device 60 and external components. For example, the CPU 62B may be configured to function as a user interface via a console or a data port (e.g. Telnet). It can also perform connection and network management for various protocols such as Simple Network Management Protocol (SNMP).

FIGURE 8 shows a block diagram of various inputs/outputs, components and connections of a system 800 which may be used for implementing various aspects of

the present invention. According to a specific embodiment, system 800 may correspond to CPU 62A of FIGURE 7.

As shown in the embodiment of FIGURE 8, system 800 includes cell switching logic 810 which operates in conjunction with a scheduler 806. In one implementation, cell switching logic 810 is configured as an ATM cell switch. In other implementations, switching logic block 810 may be configured as a packet switch, a frame relay switch, etc.

Scheduler 806 provides quality of service (QoS) shaping for switching logic 810. For example, scheduler 806 may be configured to shape the output from system 800 by controlling the rate at which data leaves an output port (measured on a per flow/connection basis). Additionally, scheduler 806 may also be configured to perform policing functions on input data. Additional details relating to switching logic 810 and scheduler 806 are described below.

As shown in the embodiment of FIGURE 8, system 800 includes logical components for performing desired format and protocol conversion of data from one type of communication protocol to another type of communication protocol. For example, the system 800 may be configured to perform conversion of frame relay frames to ATM cells and vice-versa. Such conversions are typically referred to as interworking. In one implementation, the interworking operations may be performed by Frame/Cell Conversion Logic 802 in system 800 using standardized conversion techniques as described, for example, in the following reference documents, each of which is incorporated herein by reference in its entirety for all purposes

<u>ATM Forum</u>

(1) "B-ICI Integrated Specification 2.0", af-bici-0013.003, Dec. 1995

(2) "User Network Interface (UNI) Specification 3.1", af-uni-0010.002, Sept. 1994

(3) "Utopia Level 2, v1.0", af-phy-0039.000, June 1995

(4) "A Cell-based Transmission Convergence Sublayer for Clear Channel Interfaces", af-phy-0043.000, Nov. 1995

<u>Frame Relay Forum</u>

(5) "User-To-Network Implementation Agreement (UNI)", FRF.1.2, July 2000

(6) "Frame Relay/ATM PVC Service Interworking Implementation Agreement", FRF.5, April 1995

(7) "Frame Relay/ATM PVC Service Interworking Implementation Agreement", FRF.8.1, Dec. 1994

ITU-T

(8) "B-ISDN User Network Interface - Physical Layer Interface Specification", Recommendation I.432, March 1993

(9) "B-ISDN ATM Layer Specification", Recommendation I.361, March 1993

As shown in the embodiment of FIGURE 8, system 800 may be configured to include multiple serial input ports 812 and multiple parallel input ports 814. In a specific embodiment, a serial port may be configured as an 8-bit TDM port for receiving data corresponding to a variety of different formats such as, for example, Frame Relay, raw TDM (e.g., HDLC, digitized voice), ATM, etc. In a specific embodiment, a parallel port, also referred to as a Utopia port, is configured to receive ATM data. In other embodiments, parallel ports 814 may be configured to receive data in other formats and/or protocols. For example, in a specific embodiment, ports 814 may be configured as Utopia ports which are able to receive data over comparatively high-speed interfaces, such as, for example, E3 (35 megabits/sec.) and DS3 (45 megabits/sec.).

According to a specific embodiment, incoming data arriving via one or more of the serial ports is initially processed by protocol conversion and parsing logic 804. As data is received at logic block 804, the data is demultiplexed, for example, by a TDM multiplexer (not shown). The TDM multiplexer examines the frame pulse, clock, and data, and then parses the incoming data bits into bytes and/or channels within a frame or cell. More specifically, the bits are counted to partition octets to determine where bytes and frames/cells start and end. This may be done for one or multiple incoming TDM datapaths. In a specific embodiment, the incoming data is converted and stored as sequence of bits which also include channel number and port number identifiers. In a specific embodiment, the storage device may correspond to memory 808, which may be configured, for example, as a one-stack FIFO.

According to different embodiments, data from the memory 808 is then classified, for example, as either ATM or Frame Relay data. In other preferred embodiments, other types of data formats and interfaces may be supported. Data from memory 808 may then be directed to other components, based on instructions from

processor 816 and/or on the intelligence of the receiving components. In one implementation, logic in processor 816 may identify the protocol associated with a particular data parcel, and assist in directing the memory 808 in handing off the data parcel to frame/cell conversion logic 802.

In the embodiment of FIGURE 8, frame relay/ATM interworking may be performed by interworking logic 802 which examines the content of a data frame. As commonly known to one having ordinary skill in the art of network protocol, interworking involves converting address header and other information in from one type of format to another. In a specific embodiment, interworking logic 802 may perform conversion of frames (e.g. frame relay, TDM) to ATM cells and vice versa. More specifically, logic 802 may convert HDLC frames to ATM Adaptation Layer 5 (AAL 5) protocol data units (PDUs) and vice versa. Interworking logic 802 also performs bit manipulations on the frames/cells as needed. In some instances, serial input data received at logic 802 may not have a format (e.g. streaming video), or may have a particular format (e.g., frame relay header and frame relay data).

In at least one embodiment, the frame/cell conversion logic 802 may include additional logic for performing channel grooming. In one implementation, such additional logic may include an HDLC framer configured to perform frame delineation and bit stuffing. As commonly known to one having ordinary skill in the art, channel grooming involves organizing data from different channels in to specific, logical contiguous flows. Bit stuffing typically involves the addition or removal of bits to match a particular pattern.

According to at least one embodiment, system 800 may also be configured to receive as input ATM cells via, for example, one or more Utopia input ports. In one implementation, the protocol conversion and parsing logic 804 is configured to parse incoming ATM data cells (in a manner similar to that of non-ATM data) using a Utopia multiplexer. Certain information from the parser, namely a port number, ATM data and data position number (e.g., start-of-cell bit, ATM device number) is passed to a FIFO or other memory storage 808. The cell data stored in memory 808 may then be processed for channel grooming.

In specific embodiments, the frame/cell conversion logic 802 may also include a cell processor (not shown) configured to process various data parcels, including, for

example, ATM cells and/or frame relay frames. The cell processor may also perform cell delineation and other functions similar to channel grooming functions performed for TDM frames. As commonly known in the field of ATM data transfer, a standard ATM cell contains 424 bits, of which 32 bits are used for the ATM cell header, eight bits are used for error correction, and 384 bits are used for the payload.

Once the incoming data has been processed and, if necessary, converted to ATM cells, the cells are input to switching logic 810. In a specific embodiment, switching logic 810 corresponds to a cell switch which is configured to route the input ATM data to an appropriate destination based on the ATM cell header (which may include a unique identifier, a port number and a device number or channel number, if input originally as serial data).

According to a specific embodiment, the switching logic 810 operates in conjunction with a scheduler 806. Scheduler 806 uses information from processor 816 which provides specific scheduling instructions and other information to be used by the scheduler for generating one or more output data streams. The processor 816 may perform these scheduling functions for each data stream independently. For example, the processor 816 may include a series of internal registers which are used as an information repository for specific scheduling instructions such as, expected addressing, channel index, QoS, routing, protocol identification, buffer management, interworking, network management statistics, etc.

Scheduler 806 may also be configured to synchronize output data from switching logic 810 to the various output ports, for example, to prevent overbooking of output ports. Additionally, the processor 816 may also manage memory 808 access requests from various system components such as those shown, for example, in FIGURES 7 and 8 of the drawings. In a specific embodiment, a memory arbiter (not shown) operating in conjunction with memory 808 controls routing of memory data to and from requesting clients using information stored in processor 816. In a specific embodiment, memory 808 includes DRAM, and memory arbiter is configured to handle the timing and execution of data access operations requested by various system components such as those shown, for example, in FIGURES 7 and 8 of the drawings..

Once cells are processed by switching logic 810, they are processed in a reverse manner, if necessary, by frame/cell conversion logic 802 and protocol conversion logic

804 before being released by system 800 via serial or TDM output ports 818 and/or parallel or Utopia output ports 820. According to a specific implementation, ATM cells are converted back to frames if the data was initially received as frames, whereas data received in ATM cell format may bypass the reverse processing of frame/cell conversion logic 802.

For purposes of illustration, the techniques of the present invention have been described with reference to their applications in ATM networks. However, it will be appreciated that the scheduling resource allocation technique of the present invention may be adapted to be used in a variety of different data networks utilizing different protocols such as, for example, packet-switched networks, frame relay networks, ATM networks, etc. Such modifications will be readily apparent to one having ordinary skill in the art.

Although several preferred embodiments of this invention have been described in detail herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to these precise embodiments, and that various changes and modifications may be effected therein by one skilled in the art without departing from the scope of spirit of the invention as defined in the appended claims.